# Trustworthiness Perceptions of Computer Code: A Heuristic-Systematic Processing Model

Gene M. Alarcon
Air Force Research Laboratory, OH
Wright Patterson AFB
gene.alarcon.1@us.af.mil

Tyler J. Ryan
CSRA, Dayton, OH
tyler.ryan@csra.com

## Abstract

*The popularity of code reuse and the prospect of computer generated code raises questions of how programmers trust in computer code. Psychological understanding of computer code perceptions and comprehension has yet to be explored in regards to the decision making processes involved with software development and reuse practices. A review of current literature on trust, automation, software reuse, and the intersection of the three is presented. The authors propose a theoretical model of this decision making process, building off of a heuristic-systematic model of persuasion. Future research directions and possible applications are discussed.*

## 1. Introduction

Computer code, also known as software, source code, or simply code, permeates almost every aspect of modern society. Code runs everything from power grids for cities to wearable devices that monitor heart rates. The ubiquity of code has created a need for reliable and secure code in an expeditious manner. The ability to reuse code written for another project or at a previous time has proven necessary for advancing the utility and complexity of software. A recent article by CNET estimated that approximately 80-90% of code has been reused by programmers [1]. Similarly, an analysis of projects stemming from open source developments found more than 50% of the files were used in multiple projects [2]. The reuse of code indicates a degree of trust in the code. Trusting in the code increases efficiency as time and effort can be saved by reutilizing code from previous projects or from a repository. In addition, large frameworks of code can be built by several programmers with each person reusing code from another team member to create a larger, more complex architecture. The task would be near insurmountable if one had to write a large architecture consisting of millions of lines of code by oneself. However, there are also potential risks with reusing code.

Reusing code can introduce vulnerabilities to a system. Hautala [1] indicated most of the current issues in cybersecurity are issues that were present in previous forms of code, and were reintroduced into the new program when code was reused without proper vetting. Reusing code can introduce flaws that were previously overlooked, and can introduce vulnerabilities to numerous systems because of the pervasiveness of the reuse. For example, the OpenSSL encryption banks used for online security was generally thought to be free from issues. However, Google's security team detected a serious security flaw in the software, known as heartbleed [3, 4, 5], that resulted in the theft of financial and personal data across numerous banks. The heartbleed issue was a failure to check whether a chunk of memory ended where it was supposed to end [5]. The pervasiveness of code and the risks associated with code reuse make the process of how a programmer perceives the trustworthiness of code an important aspect of psychological and computer sciences research that has rarely been explored.

The current paper develops a model of trust in code based on previous research and available psychological theories to illuminate the underlying processes that occur in trust as it relates to reuse. We begin with a brief overview of the interpersonal trust and trust in automation literatures, previous research on computer code and its reuse, and how the two interrelate. We develop a model based on the heuristic-systematic processing model, which we map to previous research. Lastly, we expand the model to new forms of code such as static and adaptive computer generated code.

## 2. Trust

Trust has been called a core social motive [6]. Research on interpersonal trust extends back to Rotter's [7] seminal research on trust. Trust, according to Rotter, is defined as the generalized expectancy that

URI: http://hdl.handle.net/10125/50560
ISBN: 978-0-9981331-1-9

good things will follow from trusting the trustee or referent. This definition was later expanded to a "willingness to be vulnerable" to another person [8]. Another aspect Mayer et al.'s [8] paper added to the trust literature is the differentiation of trust from its antecedents. Jones and Shah [9] further refine the concept of trust by differentiating trust actions, trust beliefs, and trust intentions.

Trust actions are the behaviors one performs when trusting; in other words, trust actions are the behaviors that make oneself vulnerable to another [10]. Trust is founded on a social exchange process [11] whereby trustworthy actions, if reciprocated, are met with further trust actions. Research has demonstrated trust from one individual lead to trust-based behaviors, which in turn yielded heightened trust beliefs from the other individual [12]. Trust beliefs are the perceptions the trustor has about the trustee. These beliefs include perceptions of the trustee's trustworthiness and the trustor's propensity to trust [8, 13]. Propensity to trust is an individual difference characteristic that refers to the general tendency for someone to trust other individuals [8, 14]. Propensity to trust has a global effect on one's trust intentions [13], trustworthiness beliefs [9], and trust actions [13]. However, the impact of trust propensity is most salient early in interpersonal interactions when other information may not be available [15].

Trustworthiness is the trustor's perception of the trustee or referent. These perceptions are formed as a trustor interprets and ascribes motives to the trustees' actions [16]. It is important to note these are the ascribed beliefs of the trustor and are not necessarily factual. For example, Ponzi schemes manipulate perceptions of trust beliefs to scam investors out of money [17]. All scams attempt to exploit trust beliefs such as building rapport and ensuring communication appears authentic [18]. In contrast, a trustee may be trustworthy, but the trustor inaccurately does not trust the trustee. Although the perceptions may not be accurate, they infer an information-processing model to trust actions over time [9]. As the relationship develops, more information becomes known over time and is available to guide trustworthiness beliefs. As interactions mature, the trustor increasingly depends on the behavior of the trustee rather than one's own personal dispositional factors, such as propensity to trust, when making trust evaluations [9, 19]. Lastly, trust intentions are the declared willingness to be vulnerable. It is important to note, trust intentions are differentiated from trust actions, which are the actual behaviors. For example, stating you trust someone enough to play the trust fall game (intentions) is different from actually falling backwards with the expectation of being caught (action). Trust intentions are a motive structure, whereas actions are the specific behaviors.

## 2.1 Trust in automation

The trust literature has been extended to computers and robotic systems in the last few decades. Research in the fields of automation, autonomy, and robotics have shown an increased interest in trust. Although the referent is no longer a human, trust is an important issue in system design. Trust in automation can lead to increased efficiency, allowing workers to increase productivity and simultaneously use fewer cognitive resources as workers are no longer burdened by aspects of the task [20]. Sheridan [21] (p. 77) stated, "human operator trust in automation is now a major topic of interest because it significantly affects whether and how automation is used." Trustworthiness, in automation, is important because it leads to trust [8]. Trust, in turn, leads to trust behaviors, such as relying on an automated system [22]. However, overt trust in automation is not the goal of the research in this field, but rather proper calibration.

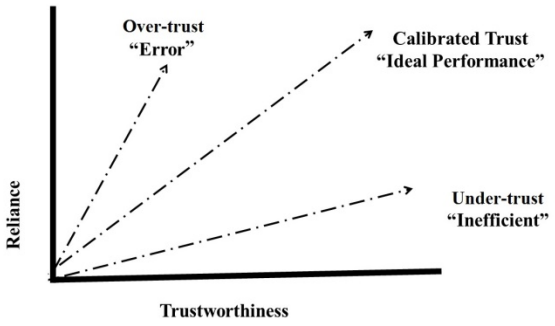


**Figure 1. Trust Calibration adapted from Lee and See [22]**

Trust calibration is an area of research that emerged in the trust in automation literature that was not prevalent in the interpersonal trust literature. Trust calibration is a function of the perceived reliability of a system and its actual reliability [22]. Figure 1 illustrates the trust calibration model. Trust calibration asserts there is an optimal level of trust between a human and an automated system. Perceiving higher reliability than the system actually affords results in over-trusting, wherein a system is relied upon when it should not be relied upon. In contrast, perceiving lower reliability than the system actually affords results in under-trusting. Trust calibration has also been studied in the computer science literature under the term

credibility. Fogg and Tseng [23] discussed the four evaluations of credibility, which follow the same general rules as trust calibration. Instead of over-trust and under-trust, the terms are gullibility error and incredulity error, respectively. This research was in reference to most any interaction with computers.

A key aspect to trust in automation is transparency. Transparency is the amount of information the system conveys to the user about how the system has come to the decision [22]. Research has demonstrated the influence of transparency on trust calibration. Lyons [24] found increased transparency on the automatic ground collision avoidance system (Auto-GCAS) greatly increased trust in the system and the use of the system. Transparency has also been related to trustworthiness perceptions in websites, such that websites that were more readable and easily accessible were viewed as higher in trustworthiness [25].

The trust in automation literature deals mainly with the interfaces a user has with a system. The appropriate level of transparency depends on the user's requirements. Research on systems such as the auto-GCAS has explored transparency in the end user [24]. However, transparency requirements are different depending on how one interacts with the systems. For example, software engineers also interact with the auto-GCAS when they update the system or install patches. These interactions require different levels of transparency [26].

## 3. Computer Code and Code Reuse

Software, or code, is a set of computer commands written in a language discernable to humans. Ultimately, all modern code is translated to machine language or machine code, which consists of binary inputs, via compilers. Programming has become increasingly complex over the past few decades as languages have evolved. High-level programming languages, such as Java, C++, and C#, are designed to make programs easier to write and modify than writing in machine language. Although high-level programming enables ease of writing and modification of the code, code can get complex and confusing. Depending on the language the code was written in, code is comprised of functions, objects, methods, and comments.

Frakes and Kang [27] defined software reuse as, "the use of existing software or software knowledge to construct new software," (p. 529). Programmers are able to forego the time and effort required to rewrite software by reusing prewritten assets, whether it be small components of code, complete packages, or libraries. Reuse has advantages and disadvantages associated with it. Research demonstrated that reusing code can increase efficiency in teams, leading to less time for code development and increasing the amount of code produced in a given time frame [28]. In addition, reusing code can lead to better code as the reused code might be more stringently attended to than newly written code, if properly vetted. However, code reuse can also have detrimental effects on overall software development. Most software flaws inherent in new programs are legacy issues as they were issues in previous code, such as security issues, indicating the code has been reused in new programs and not been properly reviewed [1].

Research on code reuse was prevalent in the 1990s, and research was focused on the non-technical aspects of code such as the team or organization. A study surveying 20 projects found increased efficiency when code reuse was implemented at the organizational level, but the study did not explore the efficiency of the code or errors that were propagated because of code reuse [28]. Lim [29] found code reuse resulted in a 24-51% reduction in defects per thousand non-comment source statements. Lim also found a 40-57% increase in overall software production when code reuse was implemented. Although organizational factors do play a role in the reuse of code, ultimately the decision occurs with an individual programmer.

Until recently, individual perceptions of the programmer have largely been discounted in the literature. Recently, a cognitive task analysis (CTA) was performed to determine the factors that comprise trustworthiness in code reuse by interviewing programmers [30]. The study found three factors that comprise trustworthiness: reputation, transparency, and performance. Reputation was defined as trustworthiness cues based on information provided outside the code, such as source, number of reviews, and number of users of the code, to name a few. Transparency was defined as the perceived comprehensiveness of the code from viewing it, aided by proper adherence to established conventions. Lastly, performance was defined as the capability of the code to meet the necessities of the project. Although these three factors comprised trustworthiness in code, individual differences and environmental factors were also posited to have an influence on trust in code. Individual differences may include aspects such as personality (e.g., propensity to trust), individual training or experience differences (e.g., formal training versus self-teaching; novice versus veteran programmer). Environmental differences are constraints placed on the programmer. These may be constraints because of the type of code (e.g., server

code versus stand alone or isolated code), organizational constraints placed on the programmer, customer needs, or customer requirements [30]. Although the study described the factors that pertain to trustworthiness assessments of the code, no cohesive theory or model was advocated for the processes by which trustworthiness of code is assessed. Next, we discuss a psychological model that is relevant to code reuse.

## 4. Heuristic-Systematic Model

The Heuristic-Systematic Model (HSM) is an information processing model in the psychological literature that was originally developed for persuasive messages [31]. The model has proven its utility in several aspects of psychological and consumer research. The model helps to elucidate both the cognitive processes and actual behaviors, thus encompassing a broader spectrum. The model may also add insight into how programmers view the trustworthiness of code and how they determine to reuse computer code. The model proposes two modes of processing information: heuristic processing and systematic processing. A heuristic is a "strategy that ignores part of the information, with the goal of making decisions quickly, frugally" (p. 454) than more intricate strategies [31]. Heuristic processing relies on judgmental rules (e.g., rules of thumb, industry conventions, best practices). Decisions that are made with heuristic processing rely on information cues rather than more in-depth analyses. Systematic processing relies on thorough analytic processing of available information. Heuristic processing is generally faster and requires fewer cognitive resources. In contrast, systematic processing takes more time and cognitive resources. Heuristic and systematic processing can occur alone or co-occur [32]. The co-occurrence can be in an additive fashion [33] or one process can bias the processing of the other process [34].

### 4.1 Sufficiency Principle.

The HSM relies on two aspects to determine the appropriate level of processing: efficiency and confidence. The HSM assumes perceivers desire to exert the least amount of effort necessary to determine a judgement, in other words efficiency [35]. Efficiency must be counter balanced with confidence in the decision. The confidence in the decision is based on other relevant motives, particularly motivation, which is discussed below. There is the actual level of the perceiver's confidence and the desired level of confidence. If actual confidence is lower than desired confidence, then processing continues. A balance between efficiency and confidence occurs when the perceiver is confident enough in the judgement that it satisfies the motives. This is known as the sufficiency threshold. If the desired confidence level is not met, then the perceiver continues processing information until the desired confidence level is met. Systematic processing is often more effective than heuristic processing in increasing subjective level of confidence, despite the time and cognitive effort it requires.

### 4.2 Motivation.

The motivation of the perceiver is also of importance in the HSM. Three types of motivation for processing are discussed in the model: accuracy, defense, and impression motivations [36]. Accuracy motivation is, "an open-minded and evenhanded treatment of judgment-relevant information" [36, p. 45]. When cognitive resources and/or motivation are low, accuracy motivation declines, which leads to heuristic processing. In contrast, when motivation and cognitive resources are high, systematic processing occurs to achieve the programmer's accuracy concerns [37]. Defense motivation is the desire to form judgements based on congruence with one's self-definitional beliefs or own self-interest. Heuristics that are affable to one's self-concept are likely to be used. When defense motivation is high and cognitive resources are available, systematic processing emerges but the systematic processing of information might be biased. Information that is congruent with one's belief is assessed more favorably and scrutinized less than information incongruent with one's self-concept. Impression motivation refers to the desire to form judgements based on social goals. Impression motivation is similar to defense motivation in that selective systematic processing of information occurs. However, the focus of impression motivation is achieving social goals rather than retaining self-concept beliefs.

## 5. Heuristic-Systematic Model of Trust in Code

Although the HSM was created for the persuasion literature, we assert that the model adequately describes how a programmer perceives trust in code and determine to reuse code. We chose the HSM as it is a model based on the information processing approach, which dictates humans process information rather than simply respond to stimuli. Indeed, programmers must cognitively process code to

understand it. Trust has been viewed as a heuristic in psychology [38] and the information sciences [39]. Trustworthiness in code is closely related to the sufficiency threshold. Trust is a balance of risk versus confidence in the referent and the sufficiency threshold is the point where one feels confident the judgement satisfies their current motives. In the current context, the judgment is trust.

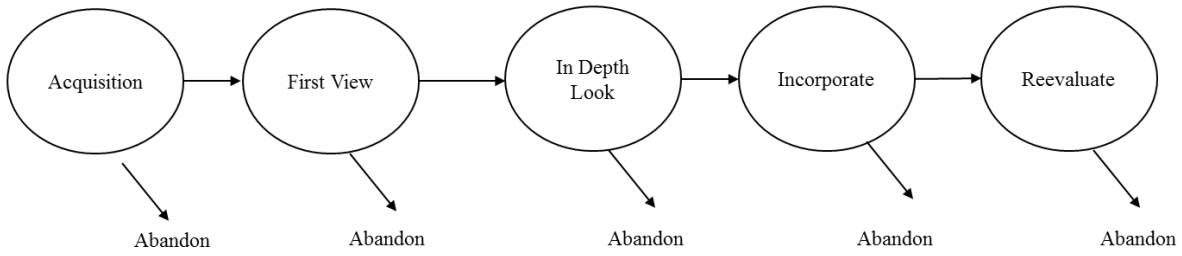


**Figure 2. Cognitive Process Model of Code Evaluation**

Perceiving someone or something as trustworthy leads to increased trust actions in human-human interactions [9] or reliance actions in human-automation interaction [22], or in the case of code, reuse in a new program. Computer programmers are taught various rules of thumb, conventions, and styles of writing code in their training [40, 41, 42]. These conventions can be thought of as heuristics. However, programmers rarely, if ever, rely simply on heuristics when evaluating code. Instead, the heuristics act as perceptions that bias further assessments of the code. Furthermore, code assessment does not occur at a single time, but rather this process is iterative, building on itself. This is in contrast to the persuasion literature, where participants demonstrated a propensity to comply due to heuristic processing [35], and the process occurs quickly at one time point. Figure 2 illustrates the process of code evaluation. In this section, we outline our Heuristic-Systematic Model of Code (HSMC) and discuss the influence of trust in each of the steps in the process.

The persuasion literature, from which the HSM is derived, operates on the key hypothesis that people attempt to conserve resources by exerting minimal effort [36]. This leads to a *select-in* approach to processing. For example, a person may perform heuristic processing by voting solely on the basis of political party. In this instance, no systematic processing occurs. Conversely, a person forming an opinion about a political candidate may *select-out*, meaning s/he does not believe the persuasive message because the persuader is from an opposing political party. However, in programming, systematic processing is necessary for evaluating code. It is difficult to imagine a time when a programmer would make a decision to trust a piece of code without reviewing it because it came from a reputable source[1]. In assessing trust in code, the select-in approach is not utilized, but rather a select-out approach to heuristic processing is performed. Programmers may abandon code based on heuristics. If code is from an unknown source, then programmers may abandon the code depending on the environment. Similarly, if issues arise in the code such as organization, readability, or other issues this may lead a programmer to abandon the code. This is often the case with "spaghetti code" which does not follow industry conventions because sections have been rewritten by several programmers or a programmer did not take the time to write the code clearly and concisely. This selecting-out heuristic of code occurs in many stages of the reuse process and may depend on different aspects of trustworthiness depending on the step in the process. Similarly, systematic processing occurs throughout the code reuse process. In the persuasion literature, to maximize efficiency one can rely on heuristics and if the threshold is met a decision can be made [35]. However, when reviewing code, programmers attempt to conserve resources by selecting out code that is not worth attempting to systematically process, as all code that is used must be systematically processed to a degree.

The authors posit code reuse at the individual level is a process with many stages. The process of reusing code has five steps: 1) acquisition, 2) initial viewing, 3) in depth look, 4) incorporation, and 5) reevaluation. First, the programmer must acquire code. This acquisition process relies heavily on the reputation factor discussed by Alarcon and colleagues [30]. Acquisition involves exploring code reviews in an

[1] Note: In the case of a programmer being mandated to use a piece of code by a manager or some other authority figure the behavior is not trust but rather obedience.

online repository such as GitHub or SourceForge. Also, programmers may receive code from a teammate or even reassess code previously written by oneself for a previous project. In this acquisition stage of the code, the actual code may not be directly viewed. For example, information can be ascertained about code on GitHub or SourceForge from reviews, comments, and the description of the code without viewing the source code. Heuristic processing is at the forefront of the initial acquisition of the code, specifically source credibility. Research in the persuasion literature has demonstrated source credibility is an important factor in decision-making [43]. Ganesan [44] identified the "reputation" of the source as a primary aspect when considering credibility in persuasion. The credibility of the source has demonstrated an effect on future processing, with participants weighing the same information from credible versus non-credible sources differently [34]. In addition, information from a non-credible source is often scrutinized more closely than information from a credible source, if systematic processing occurs [43].

Credibility of the source, called reputation [30], is typically seen as a heuristic process. In the context of peer-to-peer network exchange, the use of anonymity and pseudonyms offers opportunities for malicious applications to be spread [45]. Heuristics, such as poor reviews (i.e. number of stars, low number of downloads, etc.), influence initial perceptions of the code, even without viewing the source code, helping to avoid malicious or poorly written code. For example, Rieh and Belkin [46, 47] that found websites with domains such as .org, .gov, or .edu were perceived as more credible than .com domains, indicating heuristic processing. Similarly, when code is acquired from teammates or others, coders rely on the reputation heuristic. If one coder is handed code from a novice coder versus a veteran coder, the reputation of the programmer weighs on the assessment and decision to use the code.

Programmers rely on heuristics for evaluating the reputation of the source, but systematic processing might also occur. For example, code from a repository that has many reviews but a low rating may be processed heuristically by programmers who avoid using the code and proceed to the next option. In contrast, code that passes initial heuristic perceptions of reputation, may be examined systematically, such as determining the authors of the code or the languages it incorporates. Indeed, the primary influences of reputation assessments are at the first point of contact [48]. This all leads to a reputation assessment. Research has found code developers spend more time on code from a "Reputable" source than from an unknown source [49]. This may have led to higher trustworthiness assessments as the participants were more familiar with the code after spending more time on the code. Importantly, the perception of the reputation of the source of the code biases the assessment of the code at future time points [34]. This bias can influence the trust calibration process, leading to over or under-trust. At this point, the programmer decides to initially trust the code by downloading the code from the repository or opening the code from the teammate. This leads to the initial viewing step.

The second stage is the initial viewing step. The initial viewing step is the first time the programmer views the code. The reputation and transparency heuristics described by Alarcon and colleagues [30] play a major role in the assessment of the code at this stage. The ability to understand and comprehend source code is imperative for reusing, adapting, and debugging previously written code, which requires a human-centered approach to produce tools and best practices for future software development [50]. Code that does not adhere to stylistic practices (e.g., readable, organized, proper commenting) increases the likelihood of heuristic processing, and thus abandonment. Many programming languages have grammatical requirements necessary for a program to function, as well as stylistic rules which aid development and review of the code text [51]. Code referred to as "Spaghetti code" often violates these conventions. As such, code that violates these conventions are processed with heuristic processing, which results in abandoning the code in search of another one. Albayrak and Davenport [52] examined the effects of indentation and naming defects on code inspection. Results indicated more false positives when the code was degraded, indicating distrust. In contrast, code that generally conforms to the conventions is initially processed with heuristic processing, but after passing the heuristic processing stage, the code is further inspected. Research utilizing eye trackers on how programmers read code indicated programmers scanned the entire code quickly before looking at the code in greater depth [53].

The third stage in the process is the in-depth look. Transparency is a key aspect of trustworthiness in the in depth look step. Code that is hard to understand because of lack of transparency is seen as less trustworthy and thus reused less [30]. Research in computer credibility has postulated that aspects of websites, such as message clarity (readability and organization), presentation, and lack of mistakes, influence how participants perceive websites [38]. These factors indicate transparency of the website influences trust perceptions. In addition, studies that explored the readability and organization of code, both of which are aspects of transparency, found code that

was higher in readability but low in organization was examined longer than code that was low in readability and low in organization [49, 54], despite the code compiling. The readability and organization of a piece of code affect its accessibility and reusability by other programmers, and can help or hinder future development and maintenance efforts. A study of 1,093 Microsoft product developer discussion threads were qualitatively coded and approximately 38% of the threads contained feedback on coding conventions in general [51]. This suggests that professional developers are aware and concerned about proper style and formatting of code text. Also, the results suggest that an ample amount of effort is spent reviewing and correcting violations to conventions.

During the in depth look, the programmer may test the code. Tests often include aspects such as checking for memory leaks, data validation, and user interface issues. This testing establishes the applicability and adaptability of the code to the project, as well as resiliency and flexibility of the code, a factor found by Alarcon and colleagues [30] they called performance. Mellarkod, Appan, Jones, and Sherif [55] found the perceived usefulness of the code had a strong relationship with behavioral intentions (reuse). Performance assessments require systematic processing. The in depth problem-solving nature of performance testing necessitates systematic processing. However, previous biases about the reputation of the source of the code may influence the degree to which systematic processing occurs. Indeed, researchers found that participants who quickly scanned the code took more time to find defects [53]. Longer systematic processing occurs when the source is reputable; in contrast, shorter systematic processing occurs when the source is less reputable, as code abandonment is prevalent if problems arise. Once testing is complete, implementation occurs.

The fourth step of the process is implementation. Implementation occurs when the referent code (code to be reused) is added to the architecture being constructed. During this process, performance-based systematic processing occurs, as well as reputation-based heuristic processing. Similar to the incorporation step, when code is implemented into the larger architecture, the programmer reviews the entire code for bugs and errors. The implementation step of the process is an intention to trust the code, as the code is reused in the next software version. This step is similar to testing; however, the focus is on how the code integrates with the rest of the code for the program being developed. Previous trust assessments of the code will influence the incorporation phase, as trust has transitioned from the perceiver to the referent [9], in this case the source code. In other words, if problems arise when implementing the code, rather than abandon, the programmer may look into modifying the other components in the architecture to resolve any issues, as the programmer is familiar with the code now. After testing, the product then goes live.

The last step is reevaluation. Code is reevaluated after some unspecified time has passed. This reevaluation may be from customers about the code who use the software produced or through beta testing (sending code to real-world customers for trial). In this instance, the reputation of the source may no longer have an impact on trust in the code. Instead, trust in the code is dependent almost solely on the transparency and performance of the code. This dissolution of the influence of the individual differences is supported in the trust [9] and computer credibility literature [23, 38]. Trustworthiness loci gradually shift from individual dispositions such as propensity to trust, which is inherent in the trustor, to aspects of the trustee, or in the current case the code. Enough information is available about the referent code to make a decision about the code itself, rather than relying on the reputation of code or individual differences. Indeed, in interpersonal interactions, individual differences, such as propensity to trust, do not predict trustworthiness in familiar pairs, as a trustor has enough information to make an assessment [56]. This same process is posited to occur in trust in code.

## 5.1 Individual Differences and Environment

Trustworthiness assessments of code also rely on environmental factors and individual differences. Each individual programmer may have different motivations for reusing code. As discussed in the HSM, three motivators exist for determining the sufficiency threshold the programmer is comfortable with at each step of the process. Accuracy motivation should be the most prevalent motivator in trust in code for several reasons. First, the code must compile. Aside from compiling, there are other issues. The environment where the code is being used moderates the accuracy thresholds. For example, high security code (e.g. server code), code that deals with finances, or code that deals with personally identifying information has a higher sufficiency threshold because of the accuracy motivation. In contrast, programs that have no security requirements have a lower sufficiency threshold. These environmental differences lead to differences in trust calibration. For example, Alarcon and colleagues [49] found participants abandoned server code quickly because it was from an unknown source. The security aspects of server code is a high-risk environment and the sufficiency threshold is much higher as programmers are motivated by security. Individual

differences should also influence the sufficiency threshold of accuracy motivation. For example, individuals higher in trait suspicion and conscientiousness may have higher sufficiency thresholds, regardless of the environment, because of individual motivations. In the interpersonal trust literature, factors such as propensity to trust influence trustworthiness perceptions of partners when they are unfamiliar with one another [56]. Indeed, in the computer credibility research, constructs such as need for cognition and propensity to trust have been postulated as influences on the perceptions of computer websites [23, 38]. Similarly, the constructs of individual differences and environment have been hypothesized as factors that influence trustworthiness perceptions of code, although the study did not directly manipulate these constructs [30].

## 6. Future Applications

Research on trust in code is lacking in both the psychology and the computer science literatures. The relative quick expansion of computers into everyday life over the course of the last few decades has left researchers unprepared for the ubiquity of code. In addition, interdisciplinary research, although needed, is often hard to implement. Studies on source code comprehension that analyze eye tracking data and time for review can further elucidate when heuristic or systematic processes are taking place. Experimental methodology, such as manipulating aspects of the code according to Alarcon and colleagues' [49, 54] factors will also help to understand the cognitive processes programmers perform and when they are elicited. Little research has explored the physiological covariates of how programmers perceive code. Measurements from functional near-infrared spectroscopy or electroencephalograms can help to determine how the programmer is processing the information and in what areas of the brain. Collaborations between psychologists, computer and information system scientists are necessary to fully understand the depth and complexity of source code comprehension.

The advent of computer generated and adaptive code has important implications for trust in code. Static computer generated code has been available in the field for some time, but is rarely implemented by programmers [30]. The reason for programmers not implementing the computer generated code is that they do not understand what the code does because it is written too concisely. In addition, adaptive computer generated code is in its infancy. The DARPA cyber challenge was a contest of adaptive code that repairs the code within the system without human intervention [57]. Although the computer science industry may be years away from implementing such code on a large scale, adaptive code does present issues in that the code repairs are often unconventional upon first viewing. A key task in the future will be designing adaptive computer generated code so that its modifications are readily intelligible to developers and end users. The principles discussed above in the proposed process model of code trustworthiness are easily adapted to computer generated code. Increasing transparency in code may increase trustworthiness perceptions and thus increase trust behaviors such as reuse.

## 7. Conclusion

Trustworthiness perceptions are an important aspect of programming. The current paper sought to lay out a theoretical model for assessing trust in code and reuse of code. The proposed model describes a five step process of how programmers perceive computer code. Future research should explore these steps and verify what factors influence trust at each step. We link the research that has been conducted previously back to the psychological literature through the heuristic-systematic model of code (HSMC). The HSMC may inform future empirical research into code comprehension and source code convention design.

## References


[1] L. Hautala, "Programmers are copying security flaws into your software, researchers warn," https://www.cnet.com/news/programmers-are-copying-security-flaws-into-your-software-researchers-warn/," 2015.

[2] A. Mockus, "Large-scale code reuse in open source software," *Emerging Trends in FLOSS Research and Development, 2007. FLOSS'07. First International Workshop on*, 2007.

[3] B. Grubb, "Heartbleed disclosure timeline: Who knew what and when," *Sydney Morning Herald*, 2014.

[4] Z. Durumeric, J. Kasten, D. Adrian, J. A. Halderman, M. Bailey, F. Li, ... and V. Paxson, "The matter of heartbleed," *Proceedings of the 2014 Conference on Internet Measurement Conference*, 2014, pp. 475-488.

[5] Mitre, CVE, "Heartbleed OpenSSL bug [CVE-2014-0160], https://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2014-0160," 2014.

[6] S. T. Fiske, and S. E. Taylor, *Social cognition: From brains to culture*, Sage, 2013.

[7] J. B. Rotter, "A new scale for the measurement of interpersonal trust." *Journal of Personality*, vol. 35, 1967, pp. 651-665.

[8] R. C. Mayer, H. C. Davis, and F. D. Schoorman, "An integrative model of organizational trust," *Academy of Management Review*, vol. 20, 1995, pp. 709-734.

[9] S. L. Jones, and P. P. Shah, "Diagnosing the locus of trust: A temporal perspective for trustor, trustee, and dyadic influences on perceived trustworthiness", *Journal of Applied Psychology*, vol. 101, 2016, pp. 392-414.

[10] P. M. Blau, *Exchange and Power in Social Life*. Transaction Publishers, 1964.

[11] M. M. Pillutla, D. Malhotra, and J. K. Murnighan, "Attributions of trust and the calculus of reciprocity," *Journal of Experimental Social Psychology*, vol. 39, 2003, pp. 448-455.

[12] M. A. Serva, M. A. Fuller, and R. C. Mayer, "The reciprocal nature of trust: A longitudinal study of interacting teams", *Journal of Organizational Behavior*, vol. 26, 2005, pp. 625-648.

[13] J. A. Colquitt, B. A. Scott, and J. A. LePine, "Trust, trustworthiness, and trust propensity: A meta-analytic test of their unique relationships with risk taking and job performance," *Journal of Applied Psychology*, vol. 92, 2007, pp. 909-927.

[14] J. B. Rotter, "Generalized expectancies for interpersonal trust," *American Psychologist*, vol. 26, 1971, pp. 443-452.

[15] D. H. McKnight, L. L. Cummings, and N. L. Chervany, "Initial trust formation in new organizational relationships," *Academy of Management Review*, vol. 23, 1998, pp. 473-490.

[16] D. L. Ferrin, and K. T. Dirks, "The use of rewards to increase and decrease trust: Mediating processes and differential effects," *Organization Science*, vol. 14, 2003, pp. 18-31.

[17] M. Vasek, and T. Moore, "There's No Free Lunch, Even Using Bitcoin: Tracking the Popularity and Profits of Virtual Currency Scams," *International Conference on Financial Cryptography and Data Security*, Springer Berlin Heidelberg, 2015, pp. 44-61.

[18] E. Williams, K. Muir, and A. Joinson, "Manipulating trust: Exploiting communication mechanisms and authenticity cues to deceive", In *Handbook of Deceptive Communication*, Palgrave, 2016.

[19] D. Z. Levin, E. M. Whitener, and R. Cross, "Perceived trustworthiness of knowledge sources: The moderating impact of relationship length," *Journal of Applied Psychology*, vol. 91, 2006, pp. 1163-1171.

[20] S. Lewandowsky, M. Mundy, and G. Tan, "The dynamics of trust: Comparing humans to automation," *Journal of Experimental Psychology: Applied*, vol. 6, 2000, pp. 104-123.

[21] T. B. Sheridan, *Humans and automation: System design and research issues*, John Wiley & Sons, Inc., 2002.

[22] J. D. Lee, and K. A. See, "Trust in automation: Designing for appropriate reliance," *Human Factors: The Journal of the Human Factors and Ergonomics Society*, vol. 46, 2004, pp. 50-80.

[23] B. J. Fogg, and H. Tseng, "The elements of computer credibility," *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, 1999.

[24] J. B. Lyons, N. T. Ho, W. E. Fergueson, G. G. Sadler, S. D. Cals, C. E. Richardson, and M. A. Wilins, "Trust of an automatic ground collision avoidance technology: A fighter pilot perspective," *Military Psychology*, vol. 28, 2016, pp. 271-277.

[25] T. J. Johnson, and B. K. Kaye, "Cruising is believing?: Comparing internet and traditional sources on media credibility measures." *Journalism & Mass Communication Quarterly*, vol. 75, 1998, pp. 325-340.

[26] K. B. Bennett, and J. M. Flach, *Display and interface design: Subtle science, exact art.* CRC Press, 2011.

[27] W. B. Frakes, and K. Kang, "Software reuse research: Status and future," *IEEE transactions on Software Engineering* vol. 31, 2005, pp. 529-536.

[28] R. D. Banker, and R. J. Kauffman, "Reuse and productivity in integrated computer-aided software engineering: An empirical study," *MIS Quarterly*, vol. 15, 1991, pp. 375-401.

[29] W. C. Lim, "Effects of reuse on quality, productivity, and economics," *IEEE software*, vol. 11, 1994, pp. 23-30.

[30] G. M. Alarcon, L. G. Millitello, P. Ryan, S. A. Jessup, C. S. Calhoun, and J. B. Lyons, "A descriptive model of computer code trustworthiness," *Journal of Cognitive Engineering and Decision Making*, vol. 11, 2017, pp. 107-121.

[31] G. Gigerenzer, and W. Gaissmaier, "Heuristic decision making." *Annual Review of Psychology*, vol. 62, 2011, pp. 451-482.

[32] R. E. Petty, and J. T. Cacioppo, "The elaboration likelihood model of persuasion," In *Communication and Persuasion*, Springer, 1986, pp. 1-24.

[33] D. Maheswaran, and S. Chaiken, "Promoting systematic processing in low-motivation settings: effect of incongruent

information on processing and judgment," *Journal of Personality and Social Psychology*, vol. 61, 1991, pp. 13-25.

[34] S. Chaiken, and D. Maheswaran, "Heuristic processing can bias systematic processing: Effects of source credibility, argument ambiguity, and task importance on attitude judgment," *Journal of Personality and Social Psychology*, vol. 66, 1994, pp. 460-460.

[35] S. Chaiken, "Heuristic versus systematic information processing and the use of source versus message cues in persuasion," *Journal of Personality and Social Psychology*, vol. 39, 1980, pp. 752-766.

[36] S. Chen, K. Duckworth, and S. Chaiken, "Motivated heuristic and systematic processing," *Psychological Inquiry*, vol. 10, 1999, pp. 44-49.

[37] S. Chaiken, R. Giner-Sorolla, and S. Chen, "Beyond accuracy: Defense and impression motives in heuristic and systematic information processing," In *The Psychology of Action: Linking Cognition and Motivation to Behavior*, P. M. Gollwitzer and J. A. Bargh, Guilford Press, 1996, pp 553-578.

[38] N. C. Wathen, and J. Burkell, "Believe it or not: Factors influencing credibility on the Web," *Journal of the American Society for Information Science and Technology*, vol. 53, 2002, pp. 134-144.

[39] W. Wang, & I. Benbasat, "Attributions of trust in decision support technologies: A study of recommendation agents for e-commerce," *Journal of Management Information Systems*, vol. 24, 2008, pp. 249-273.

[40] T. Gaddis, *Starting Out with Java: From Control Structures Through Objects*, Pearson, 2015.

[41] "Geotechnical Software Services, Java Programming Style Guidelines, http://geosoft.no/development/javastyle .html," 2015.

[42] "Google, Java Style Guidelines, https://google.github.io /styleguide/javaguide.html," 2014.

[43] R. E. Petty, and J. T. Cacioppo. "Involvement and persuasion: Tradition versus integration," *Psychological Bulletin*, vol. 107, 1990, pp. 367-374.

[44] S. Ganesan, "Determinants of long-term orientation in buyer-seller relationships," *The Journal of Marketing*, vol. 58, 1994, pp. 1-19.

[45] E. Damiani, D. C. di Vimercati, P. Samarati, and F. Violante, "A reputation-based approach for choosing reliable resources in peer-to-peer networks," *Proceedings of the 9th ACM conference on Computer and communications security*. ACM, 2002, pp. 207-216.

[46] S. Y. Rieh, and N. J. Belkin, "Understanding judgment of information quality and cognitive authority in the WWW," *Proceedings of the 61st Annual Meeting of the American Society for Information Science*, vol. 35, 1998, pp. 279-289.

[47] S. Y. Rieh, and N. Belkin, "Interaction on the Web: Scholars' judgement of information quality and cognitive authority." *Proceedings of the Annual Meeting of the American Society for Information Science*, vol. 37, 2000, pp. 25-38.

[48] C. C. Self, "Credibility," In *An Integrated Approach to Communication Theory and Research*, M. B. Salwen, and D. W. Stacks, Lawrence Erlbaum Associates, 1996, pp. 435-457.

[49] G. M. Alarcon, R. F. Gamble, S. A. Jessup, C. Walter, T. J. Ryan, and D. W. Wood, "Effects of Source, Readability and Organization on Code Reuse and Trust," Manuscript in preparation.

[50] M. Spichkova, H. Liu, M. Laali, and H. W. Schmidt, "Human factors in software reliability engineering," *arXiv preprint arXiv:1503.03584*, 2015.

[51] M. Allamanis, E. T. Barr, C. Bird, and C. Sutton, "Learning natural coding conventions." *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2014, pp. 281-293.

[52] Ö. Albayrak, and D. Davenport, "Impact of maintainability defects on code inspections," *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, vol. 50, 2010, pp. 1-4.

[53] B. Sharif, M. Falcone, and J. I. Maletic, "An eye-tracking study on the role of scan time in finding source code defects," *Proceedings of the Symposium on Eye Tracking Research and Applications*, 2012, pp. 381-384.

[54] Walter, Charles, R. Gamble, Alarcon, G. M., S. Jessup, and C. Calhoun, "Developing a mechanism to study code trustworthiness," *Proceedings of the 50th Hawaii International Conference on System Sciences*, 2017, pp. 5817-5826.

[55] V. Mellarkod, R. Appan, D. R. Jones, and K. Sherif, "A multi-level analysis of factors affecting software developers' intention to reuse software assets: An empirical investigation," *Information & Management*, vol. 44, 2007, pp. 613-625.

[56] G. M. Alarcon, J. B. Lyons, and J. C. Christensen, "The effect of propensity to trust and familiarity on perceptions of trustworthiness over time," *Personality and Individual Differences*, vol. 94, 2016, pp. 309-315.

[57] "DARPA Celebrates Cyber Grand Challenge Winners, http://www.darpa.mil/news-events/2016-08-05a," 2016.